Publishing relational data as XML

Federico Ulliana
UM, LIRMM, INRIA-GraphIK

Slides collected from James Cheney

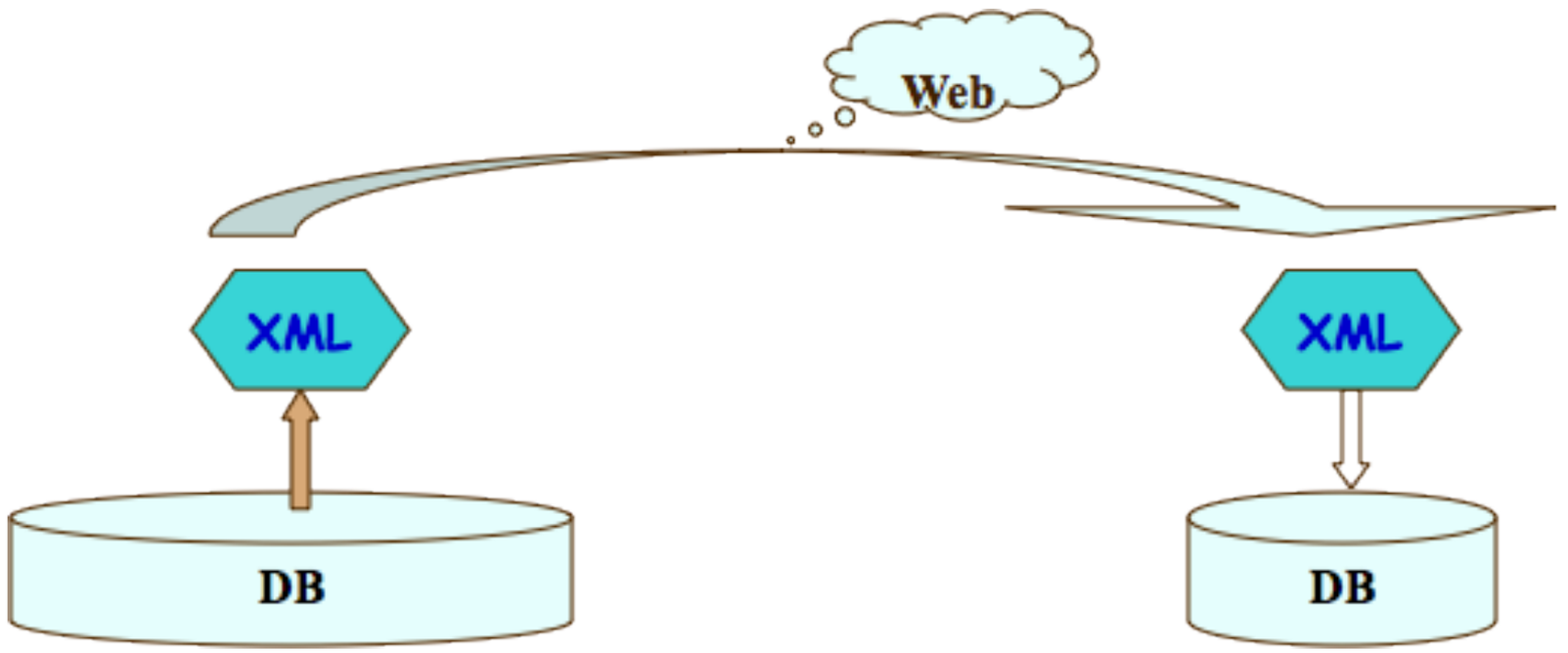
Winter ’99 : XML standardization

Jan 2000 : people (survived from millennium bug)
(and still) wondering …

Now, how can I publish online my relational data?

XML publishing

- Export relational data sources into XML



Multiple possible exports

Actors

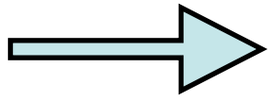
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2



```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 <Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```

Grouped by Movie

Which one to chose ?

Actors

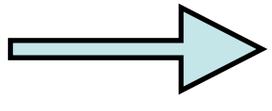
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2



```
<Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</Fname>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
<Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
  <Movie id="11">
     <Title>Spider-Man</Title>
     <Year>2002</Year>
  </Movie>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
```

Grouped by Actor

Commercial systems

Systems:

- Oracle 10g XML SQL facilities: SQL/XML
- IBM DB2 XML Extender: SQL/XML, DAD
- Microsoft SQL Server 2005: FOR-XML, XSD

Canonical XML representation of relations

- Incapable of expressing practical XML publishing
  - default fixed XML document template

Canonical XML publishing

Actors

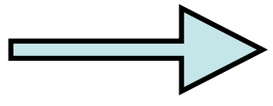
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

```
<Actors>
    <Actor aid="1">
        <LName>Maguire</LName>
        <FName>Tobey</FName>
    </Actor>
    <Actor aid="2">
        <LName>Dunst</LName>
        <FName>Kirsten</FName>
    </Actor>
</Actors>
```

Canonical XML publishing

Actors

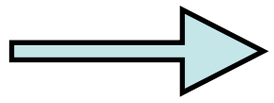
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten



```
<Actor aid="1">
 <LName>Maguire</LName>
 <FName>Tobey</FName>
</Actor>
<Actor aid="2">
 <LName>Dunst</LName>
 <FName>Kirsten</FName>
</Actor>
```

Movies

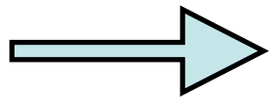
mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005



```
<Movie mid="11">
 <Title>Spider-Man</title>
 <Year>2002</Year>
</Movie>
<Movie mid="32">
 <Title>Elizabethtown</title>
 <Year>2005</Year>
</Movie>
```

Appears

mid	aid
11	1
11	2
32	2



```
<Appears mid="11" aid="1"/>
<Appears mid="11" aid="2"/>
<Appears mid="32" aid="2"/>
```

Called **canonical** because the same rules are applied to convert any relational table to an XML view

How to go beyond canonical publishing ?

Need language to specify relational-to-XML conversion

And an efficient implementation

We will see two approaches

- XPERANTO
- SilkRoute

XPERANTO

XPERANTO

[Shanmagusundaram et al., 2001]

Commercial system: IBM DB2 XML extender, SQL/XML

SQL extension

```
select    XMLAGG
from      R1, . . ., Rn
  where   conditions
```

XMLAGG Input : tables Output : XML trees (forest)

Idea: Reuse SQL Correlative Queries

Inner-query that uses variables from outer-query

- all movies with at least one actor

```
SELECT mid

FROM MOVIES M

WHERE EXISTS (      SELECT aid

                    FROM APPEARS A

                    WHERE A.mid = M.mId    )
```

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

Idea: Reuse SQL Correlative Queries

Inner-query that uses variables from outer-query

- all movies with at least one actor

```
SELECT mid
FROM MOVIES M
WHERE EXISTS (     SELECT aid
                   FROM APPEARS A
                   WHERE A.mid = M.mId    )
```

Correlative Sub-query

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

XPERANTO (SQL/XML)

Actors

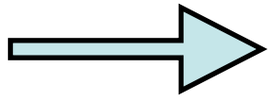
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2



```
<Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</Fname>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
<Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
  <Movie id="11">
     <Title>Spider-Man</Title>
     <Year>2002</Year>
  </Movie>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
```

Export grouping by actor

Step 1 : Define XML records

```
DEFINE XML CONSTRUCT MOVIE
(title : varchar(100),
 year : integer ) as
  {  <movie>
        <title>$title</title>
        <year>$year</year>
     </movie>  }

DEFINE XML CONSTRUCT ACTOR
(fname : varchar(100),
 lname : varchar(100),
 movie : xml ) as
  {  <actor>
        <fname>$fname</fname>
        <lname>$lname</lname>
            $movie
     </actor>  }
```

Target XML Data

```
<Actor>
        <LName>Maguire</LName>
        <FName>Tobey</FName>
        <Movie>
              <Title>Spider-Man</Title>
              <Year>2002</Year>
         </Movie>
</Actor>
...
```

Step 1 : Define XML records

```
DEFINE XML CONSTRUCT MOVIE
(title : varchar(100),
 year : integer ) as
  {  <movie>
        <title>$title</title>
        <year>$year</year>
     </movie>  }
```

```
MOVIE ('spiderman',2002) =

<movie>
      <title>spiderman</title>
      <year>2002</year>
</movie>
```

```
DEFINE XML CONSTRUCT ACTOR
(fname : varchar(100),
 lname : varchar(100),
 movie : xml ) as
  {  <actor>
        <fname>$fname</fname>
        <lname>$lname</lname>
            $movie
     </actor>  }
```

```
ACTOR('Maguyre','Tobey',<m/>)=

  <actor>
        <fname>Maguyre</fname>
         <lname>Tobey</lname>
             <m/>
  </actor>
```

Step 2 : Define Tree Aggregation

XMLAGG(<my/>,<XML/>,<trees/> **) =**

<my/><XML/><trees/>

Ready to define a SQL/XML Query

```
SELEC XMLAGG(

    ACTOR(SELECT lname, fname,

          ( SELECT XMLAGG(

              MOVIE( SELECT title , year

                     FROM Appears I1, Movies I2

                     WHERE I1.aid = O.aid

                     AND   I1.mid = I2.mid )) )

 FROM Actor  O

ORDER BY     lname, fname ) )
```

Ready to define a SQL/XML Query

```
SELEC XMLAGG(

    ACTOR(SELECT lname, fname,

          ( SELECT XMLAGG(

              MOVIE( SELECT title , year

                     FROM Appears I1, Movies I2

                     WHERE I1.aid = O.aid

                     AND   I1.mid = I2.mid )) )

 FROM Actor  O

ORDER BY     lname, fname ) )
```

Ready to define a SQL/XML Query

```
SELEC XMLAGG(

    ACTOR(SELECT lname, fname,

          ( SELECT XMLAGG(

              MOVIE( SELECT title , year

                     FROM Appears I1, Movies I2

                     WHERE I1.aid = O.aid

                     AND   I1.mid = I2.mid )) )

 FROM Actor  O

ORDER BY  lname, fname ) )
```

Ready to define a SQL/XML Query

```
SELEC XMLAGG(

    ACTOR(SELECT lname, fname,

          ( SELECT XMLAGG(

              MOVIE( SELECT title , year

                     FROM Appears I1, Movies I2

                     WHERE I1.aid = O.aid

                     AND   I1.mid = I2.mid )) )

 FROM Actor  O

ORDER BY  lname, fname ) )
```

From relations to XML Views

Actors

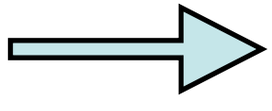
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2



```
<Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</Fname>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
<Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
  <Movie id="11">
     <Title>Spider-Man</Title>
     <Year>2002</Year>
  </Movie>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
```

Another example

Another example

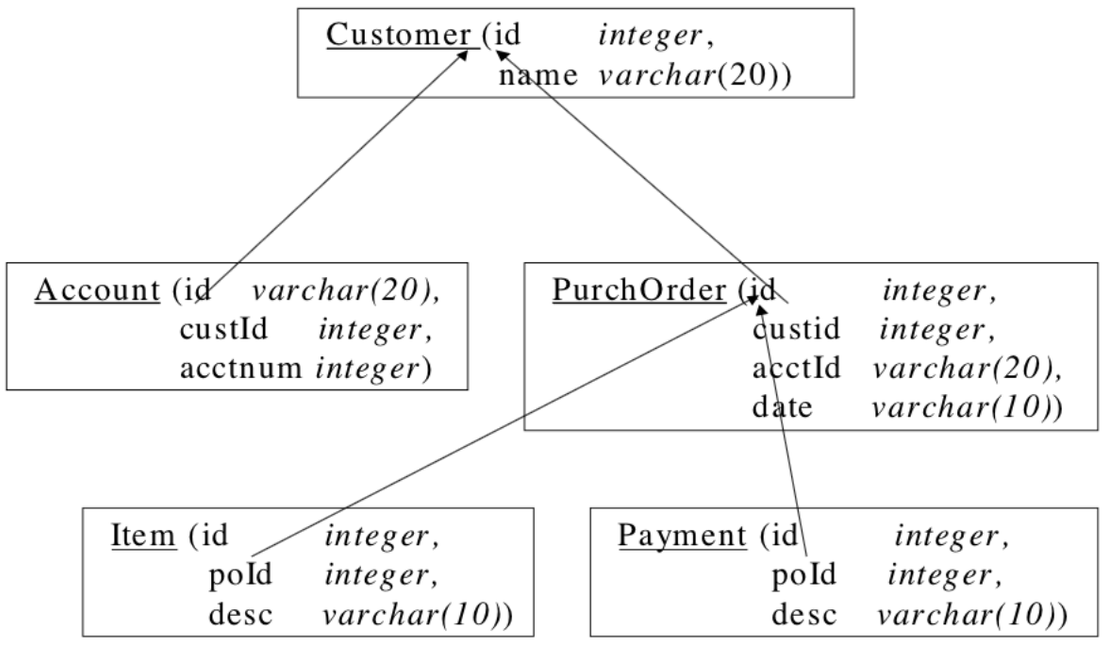


Figure 2: Customer Relational Schema

```
Define XML Constructor CUST (custId: integer,
                             custName: varchar(20),
                             acctList: xml,
                             porderList: xml) AS {
        <customer id=$custId>
                <name> $custName </name>
                <accounts> $acctList </accounts>
                <porders> $porderList </porders>
        </customer>
}
```

Figure 4: Definition of an XML Constructor

```
01. Select  cust.name, CUST(cust.id, cust.name,
02.                         (Select  XMLAGG(ACCT(acct.id, acct.acctnum))
03.                          From   Account acct
04.                          Where acct.custId = cust.id),
05.                         (Select   XMLAGG(PORDER(porder.id, porder.acct, porder.date,
06.                                                 (Select   XMLAGG(ITEM(item.id, item.desc))
07.                                                  From   Item item
08.                                                  Where item.poId = porder.id),
09.                                                (Select  XMLAGG(PAYMENT(pay.id, pay.desc))
10.                                                   From   Payment pay
11.                                                   Where pay.poId = porder.id)))
12.                          From    PurchOrder porder
13.                          Where  porder.custId = cust.id))
14. From  Customer cust
```

XPERANTO

Extends the expressive power of SQL to deal with XML

Allows to reuse existing APIs and processing infrastructure of a relational database

Contributed to the standardization of SQL'03

Implementing XML/SQL I : using stored procedures

```
FOR EACH actor a FROM Q_actor
  FOR EACH movie m FROM Q_movie(a)
      FOR EACH title t FROM Q_title(m)
      FOR EACH year  y FROM Q_year(m)
          .. BUILD-XML ..
```

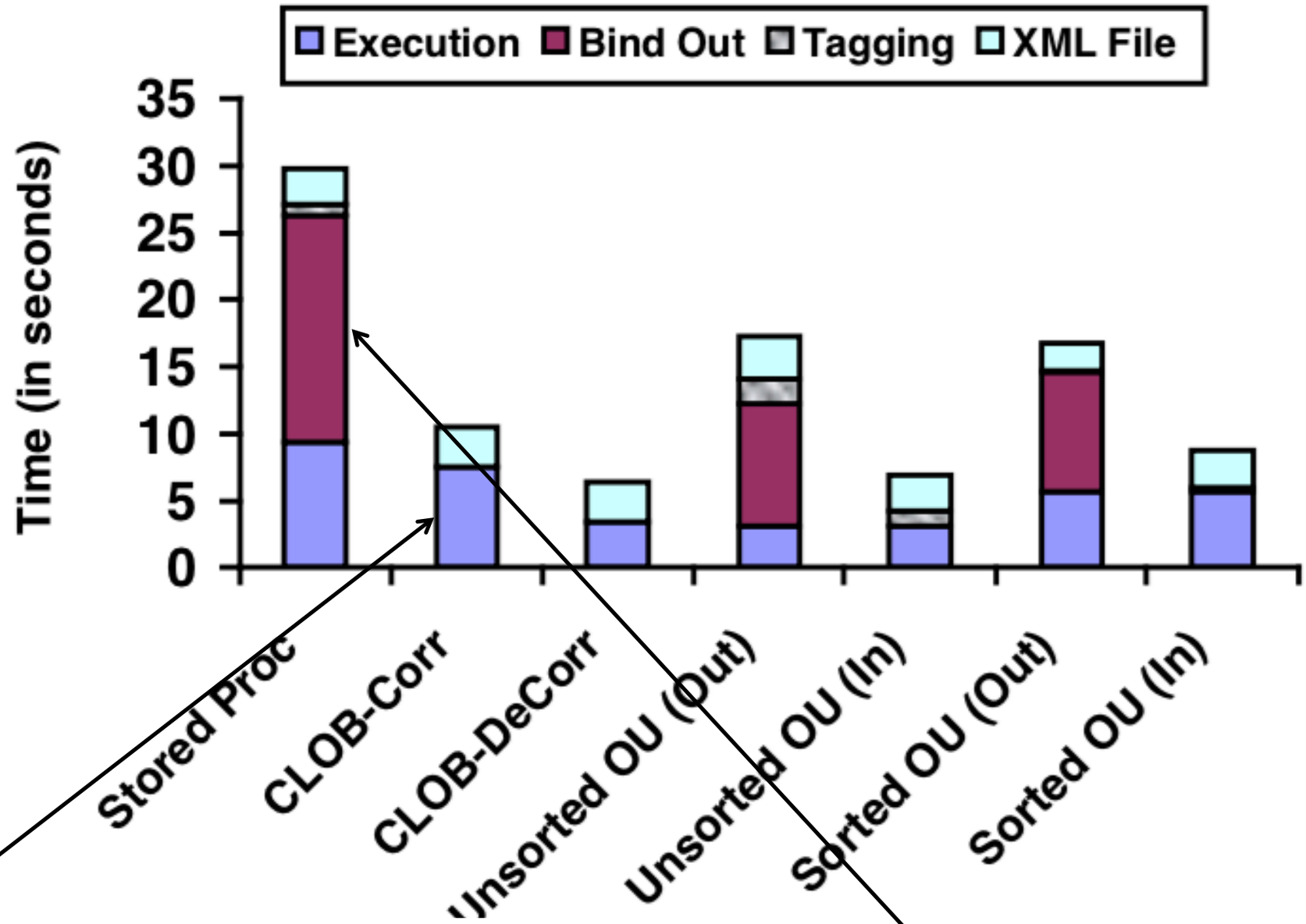
Nested-loop processing ‘outside’ the engine

No need to extend the DB engine

Implementing XML/SQL 2 : CLOBCorr(elative)

- Extend standard correlated query evaluation to add **tags** and **structure**
  - approach 'Inside' engine
- Partial XML-results stored as CLOB because of their size

Performace comparison



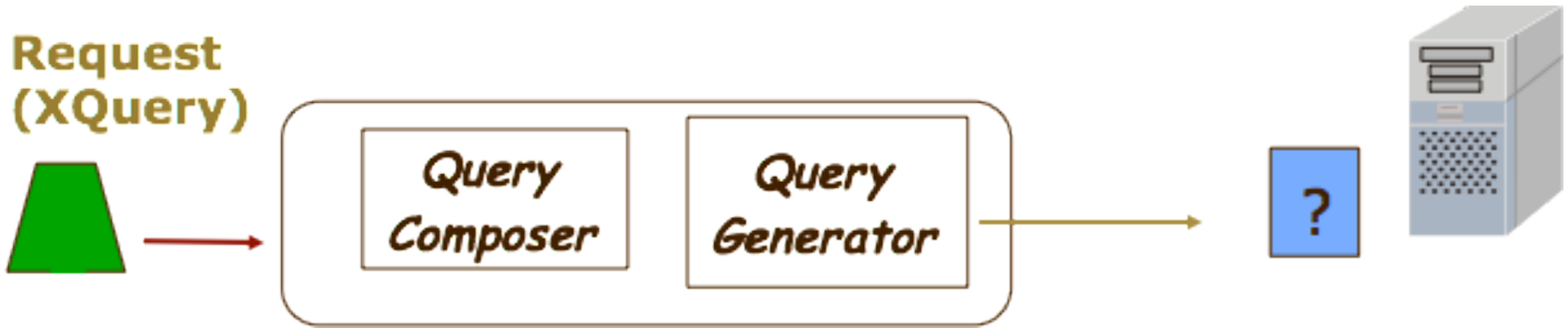
'Inside' the engine better than 'outside' because of no bind-out (=assign partial query results to variables)

SILKROUTE

SilkRoute

[Fernandez et al. 2002]

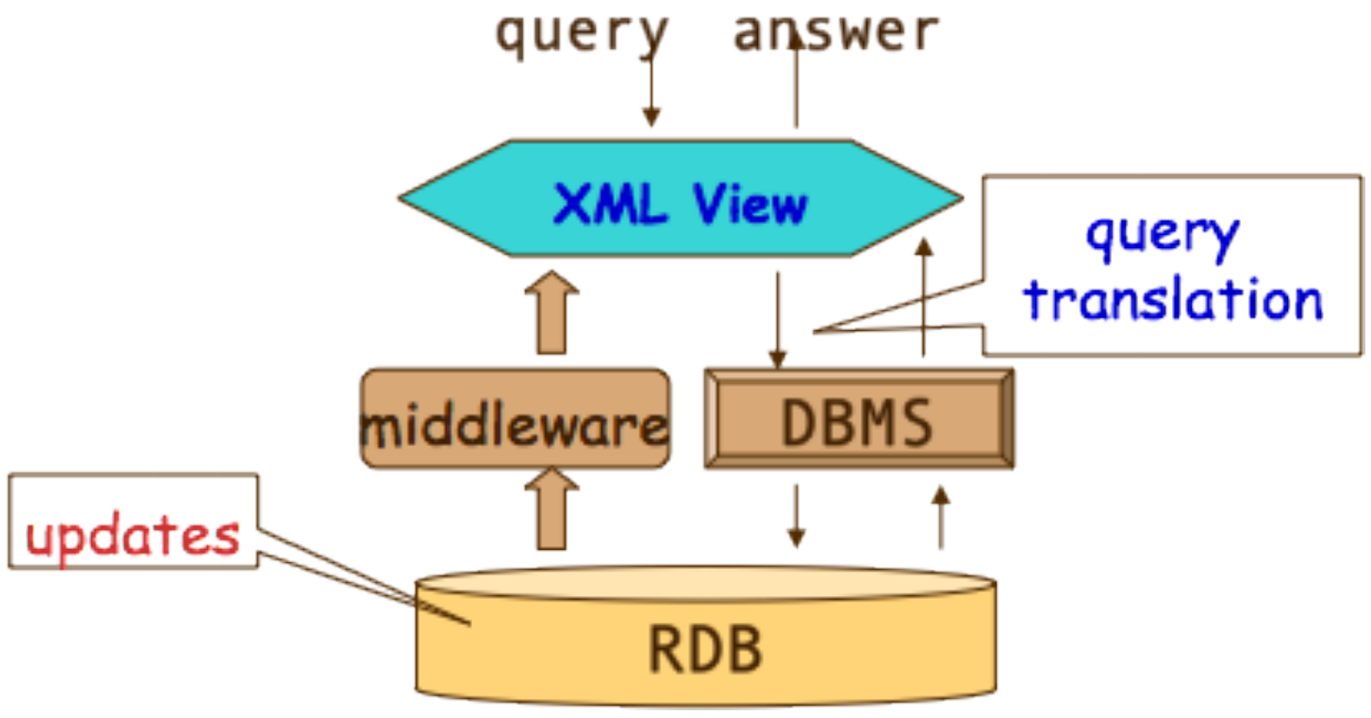
Declarative approach to query relational data based on XQuery syntax



We Go From Relations To XML

We are given the relational schema

We choose the final XML structure



Start From Relational Schema

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

Actors(id,lname,fname)

Movies(mid,title,year)

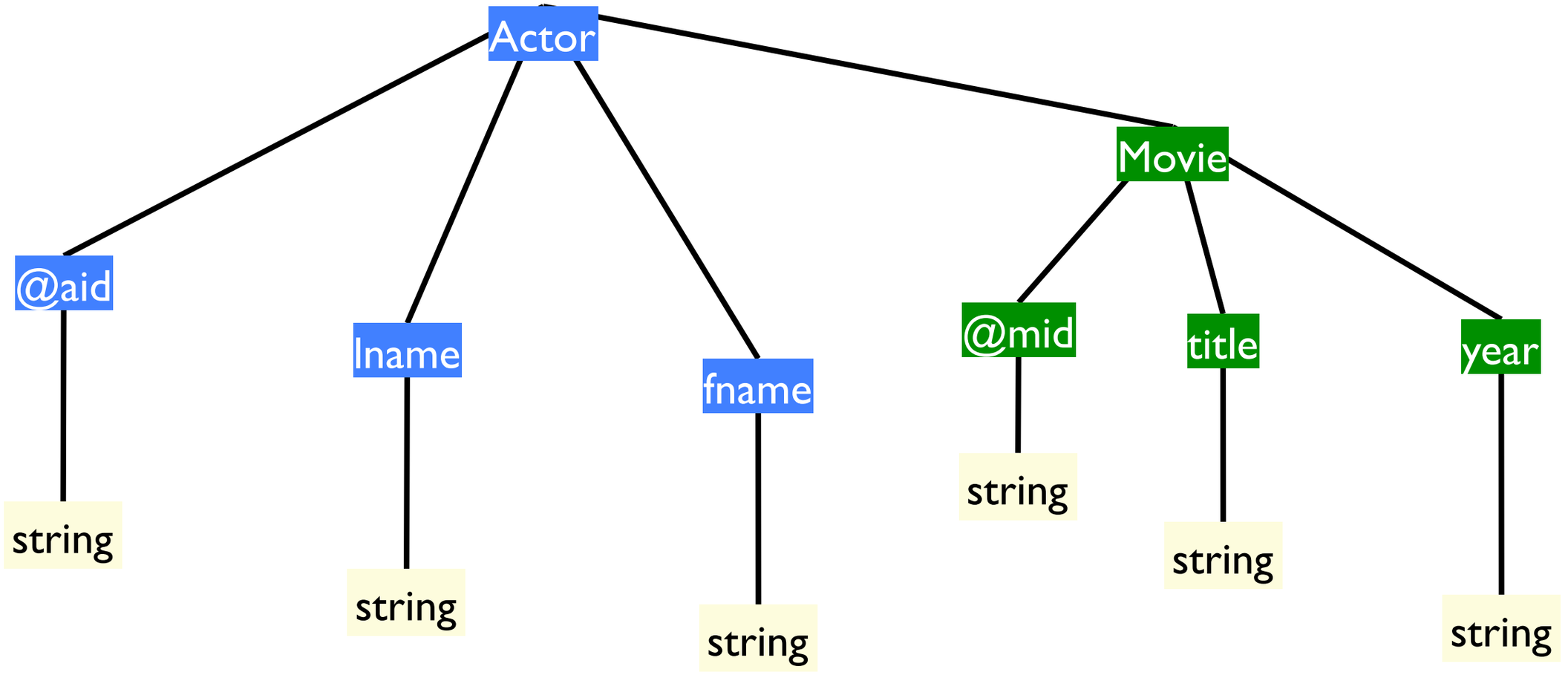
Appear(mid,aid)

First Thing : Define DTD

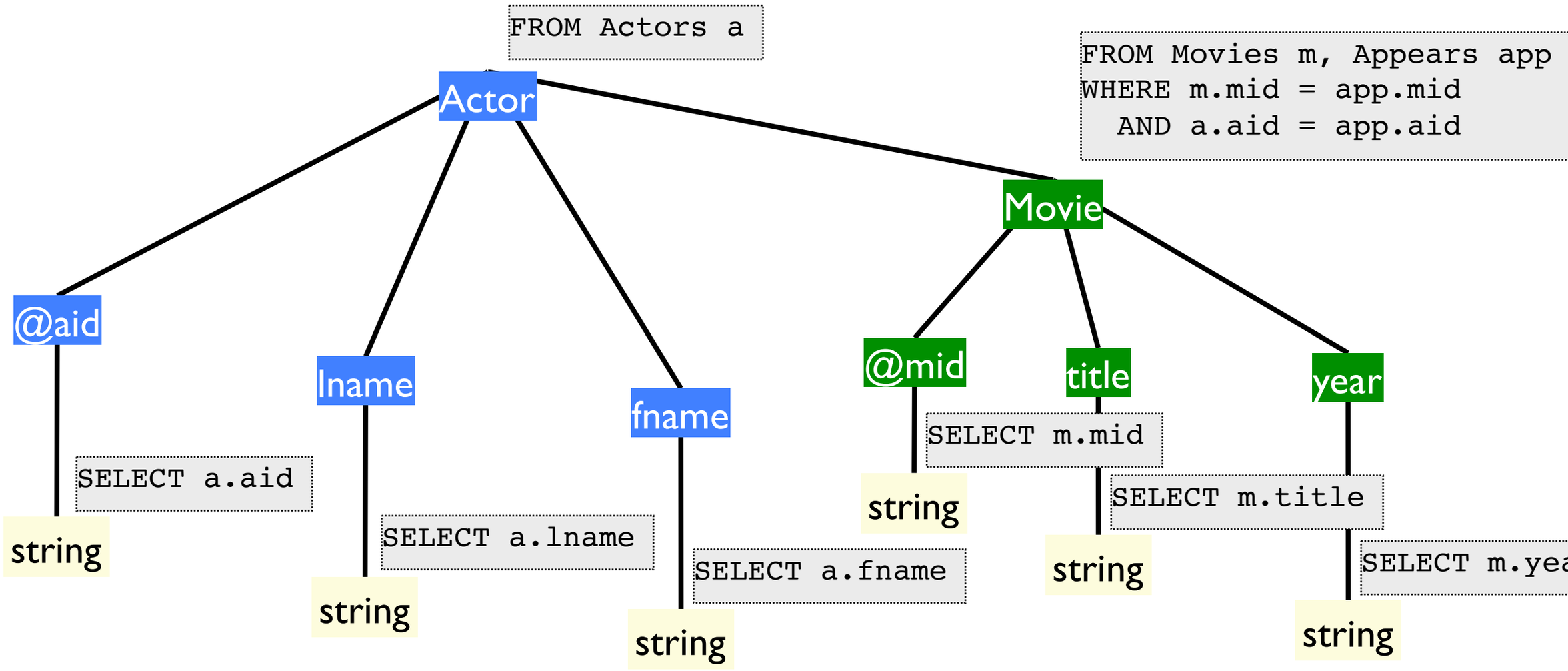
```
<!ELEMENT publicView (Actor)*>

<!ELEMENT Actor (LName,FName,Movie*)>

<!ELEMENT Movie (Titre,Year)>

<!ATTRIBUTE Movie id ID>

<!ATTRIBUTE Actor id ID>

(: all other elements are PCDATA :)
```

schema must be acyclic !

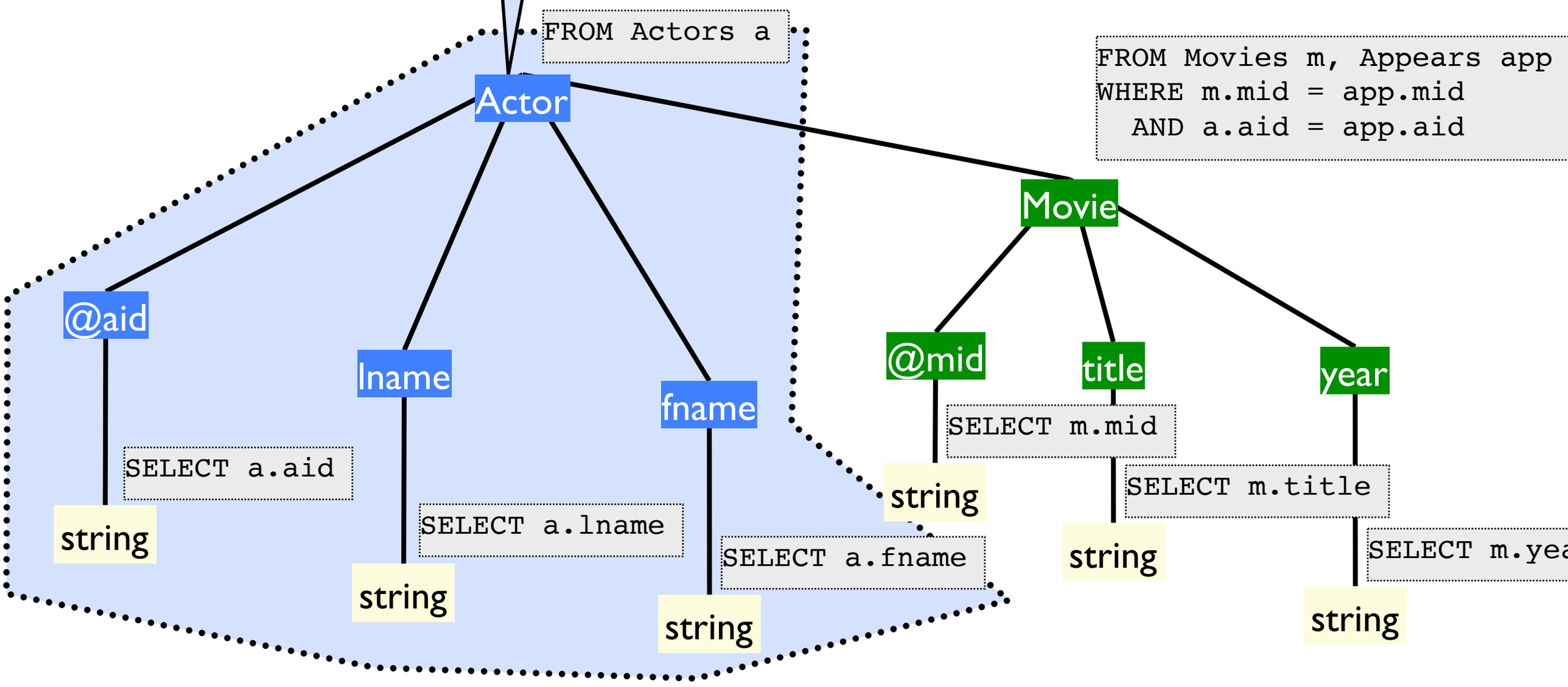
Compute Structural Tree Associated To The (Acyclic) DTD



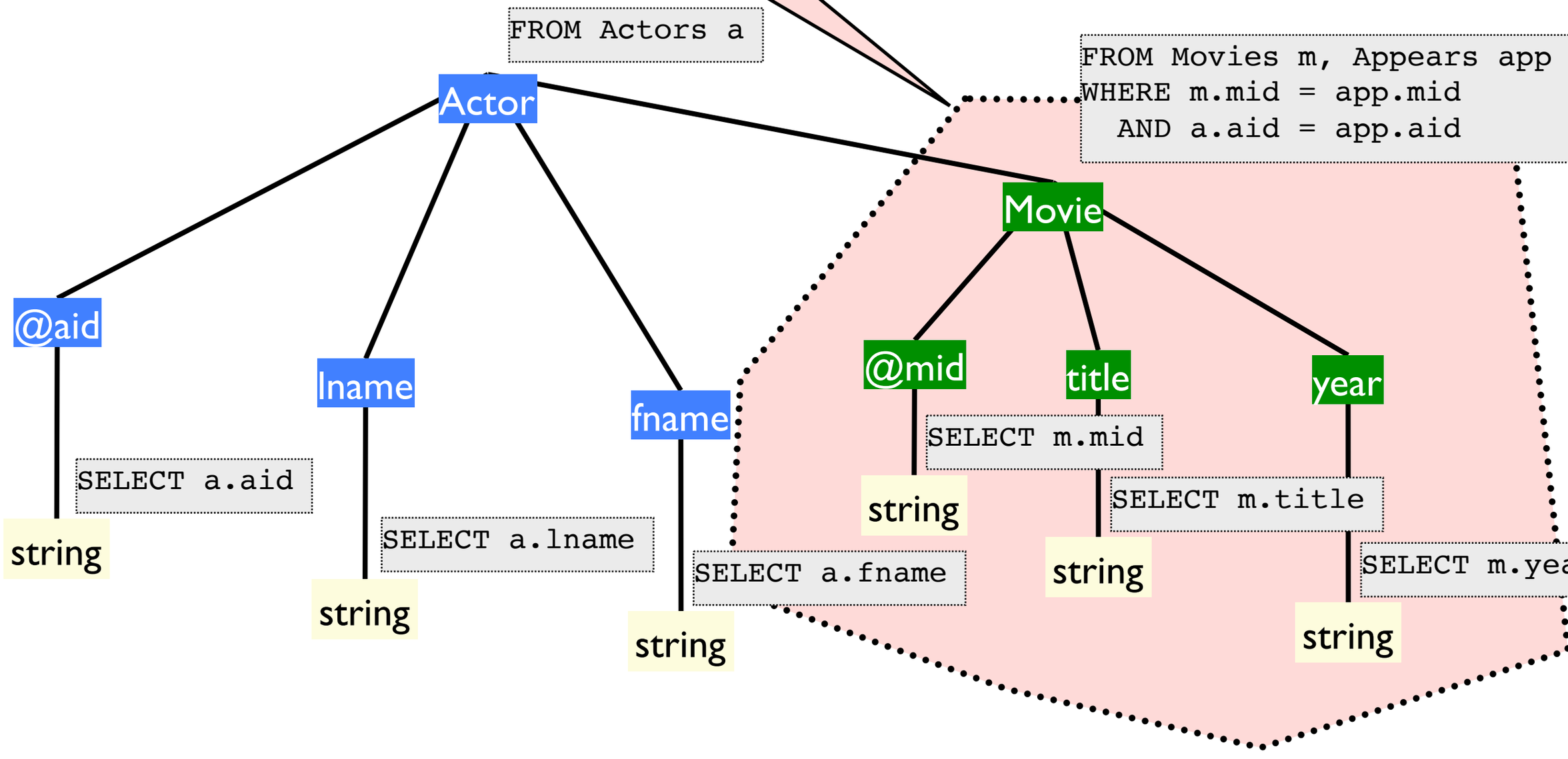
Annotate it With SQL Instructions !



```
SELECT a.aid, a.lname, a.fname
    FROM   Actors a
```



```
SELECT a.aid, m.mid, m.title, m.year
FROM   Actors a, Movies m, Appears app
WHERE  m.mid = app.mid
AND    a.aid = app.aid
```



Generate the Public View of Data

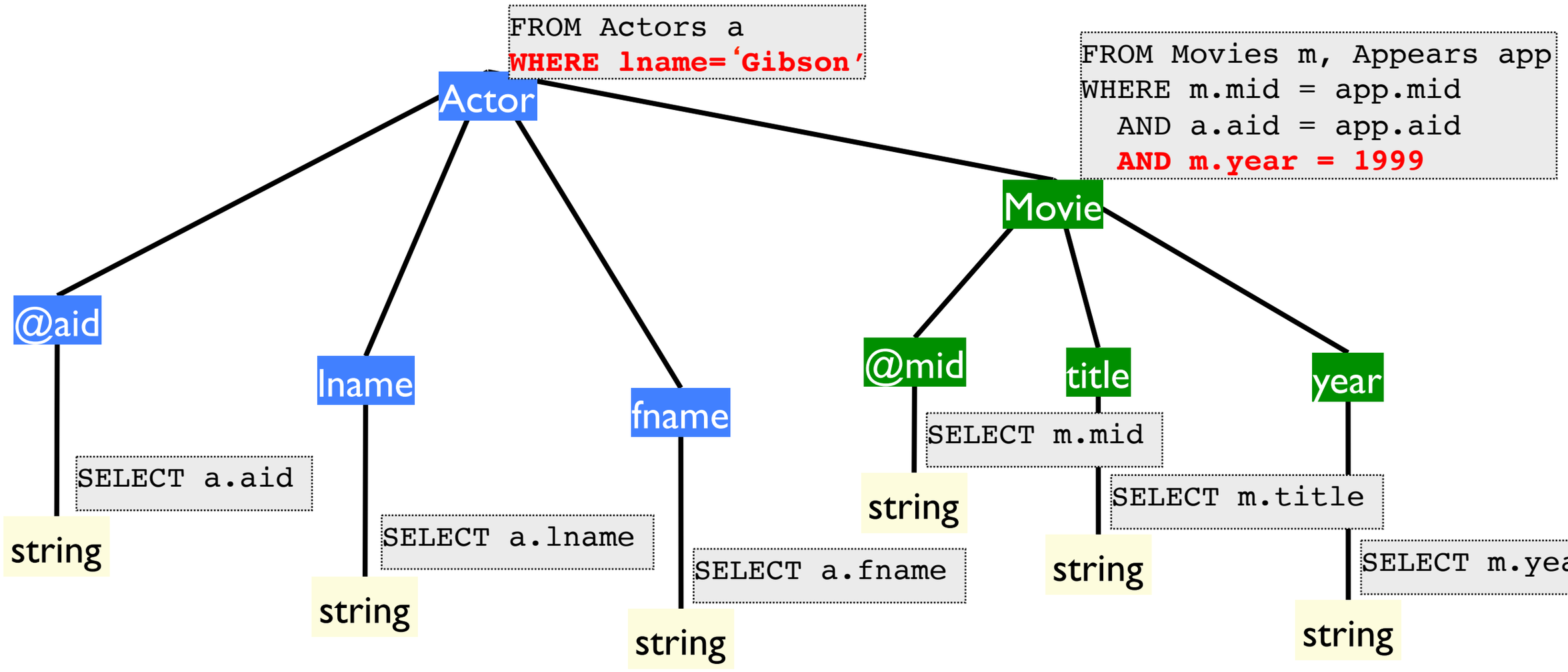
```
<publicView>

  <Actor id="1">
     <LName>Maguire</LName>
     <FName>Tobey</Fname>
     <Movie id="32">
         <Title>Elizabethtown</Title>
         <Year>1999</Year>
     </Movie>
  </Actor>

(:more actors below:)

</publicView>
```

Now, The User Comes Into Play

Q_{user}=

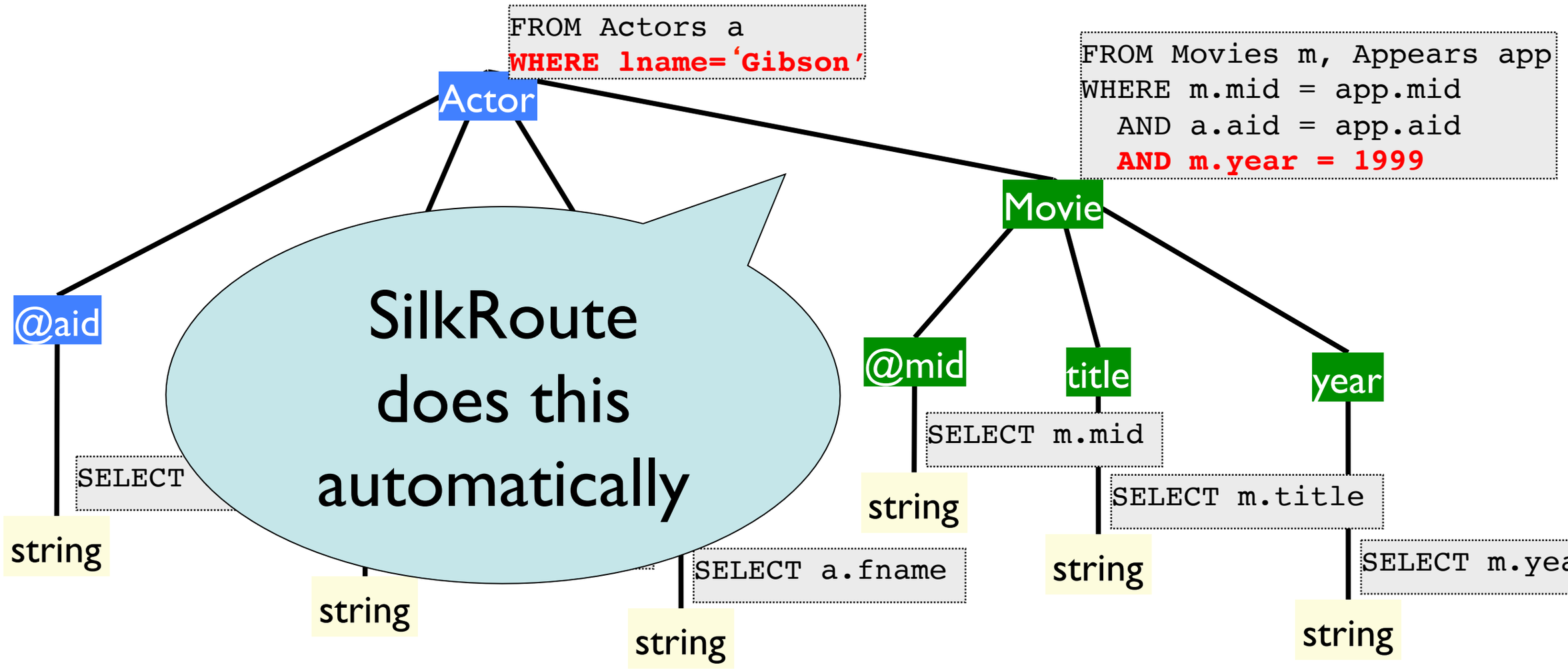
```
for $a := $publicView//Actor,
for $m := $a//Movie
where (    $a/lname='Gibson'
       and $m/year=1999    )
return $m
```

SilkRoute's SQL Generation



Query the movie of Gibson in 1999

SilkRoute's SQL Generation



Query the movie of Gibson in 1999

SilkRoute : What We Skipped

- Canonical and Public View Definition
- Translation of XQuery Expressions in Structural-trees
- Normalization of XQuery Expresssions
- Composition of XQuery Expressions
  (User Query with Public View)

Canonical XML

Actors

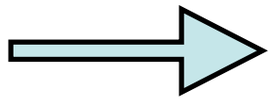
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

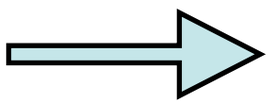
Movies

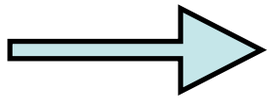
mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2







```
<canonicalView>
<Actor aid="1">
 <LName>Maguire</LName>
 <FName>Tobey</FName>
</Actor>
<Actor aid="2">
 <LName>Dunst</LName>
 <FName>Kirsten</FName>
</Actor>

<Movie mid="11">
 <Title>Spider-Man</title>
 <Year>2002</Year>
</Movie>
<Movie mid="32">
 <Title>Elizabethtown</title>
 <Year>2005</Year>
</Movie>

<Appears mid="11" aid="1"/>
<Appears mid="11" aid="2"/>
<Appears mid="32" aid="2"/>
</canonicalView>
```

Public View Declaration

```
let $cv:=//canonicalView
return
<publicView>{

for $a in $cv/Actor
return
<actor>{(
$a/LName, $a/FName,

for $m in $cv/Movie
 where (
$cv/Appears[ @aid=$a/@aid and
             @mid=$m/@mid ] )

return $m
)}<actor>
</publicView>
```

```
<canonicalView>
<Actor aid="1">
 <LName>Maguire</LName>
 <FName>Tobey</FName>
</Actor>
<Actor aid="2">
 <LName>Dunst</LName>
 <FName>Kirsten</FName>
</Actor>

<Movie mid="11">
 <Title>Spider-Man</title>
 <Year>2002</Year>
</Movie>
<Movie mid="32">
 <Title>Elizabethtown</title>
 <Year>2005</Year>
</Movie>

<Appears mid="11" aid="1"/>
<Appears mid="11" aid="2"/>
<Appears mid="32" aid="2"/>
</canonicalView>
```

Public View Declaration

```
let $cv:=//canonicalView
return
<publicView>{

for $a in $cv/Actor
return
<actor>{(
$a/LName, $a/FName,

for $m in $cv/Movie
 where (
$cv/Appears[ @aid=$a/@aid and
             @mid=$m/@mid ] )

return $m
)}<actor>
}</publicView>
```

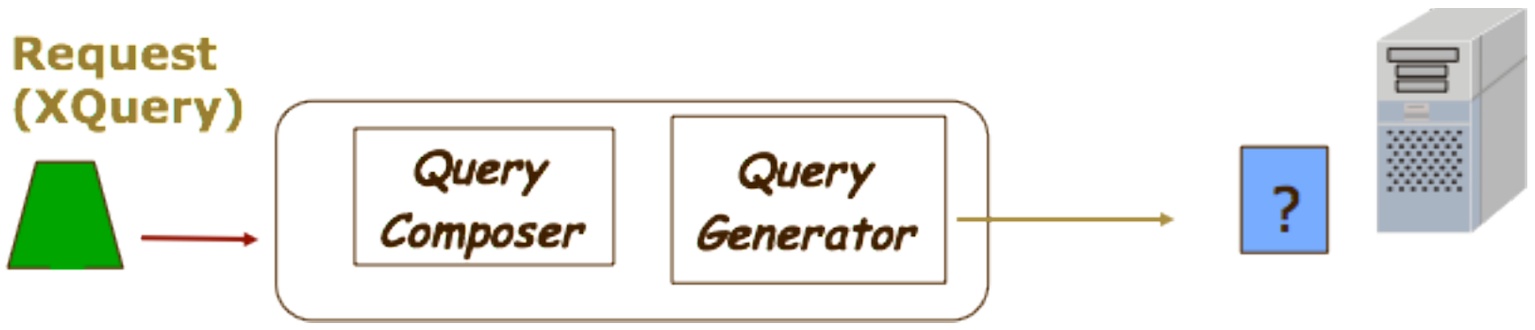
```
<publicView>
<Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</Fname>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
<Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
  <Movie id="11">
     <Title>Spider-Man</Title>
     <Year>2002</Year>
  </Movie>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
</publicView>
```

User Query

Q_{user} =

```
let $pv:= $publicView
return (: do something :)
```

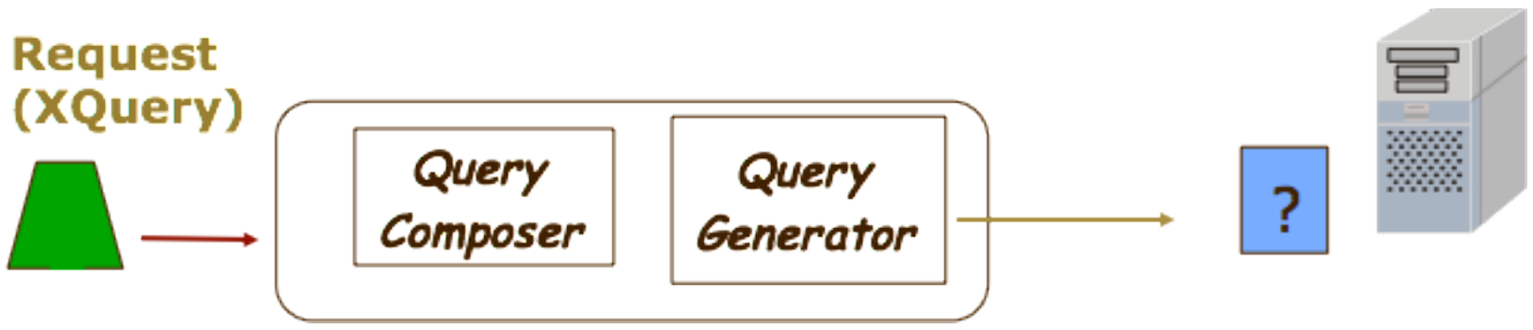
Query Composition



Q_{admin} = let $**cv**:= $**canonicalView**
return (: do something :)

Q_{user} = let $**pv**:= $**publicView**
return (: do something :)

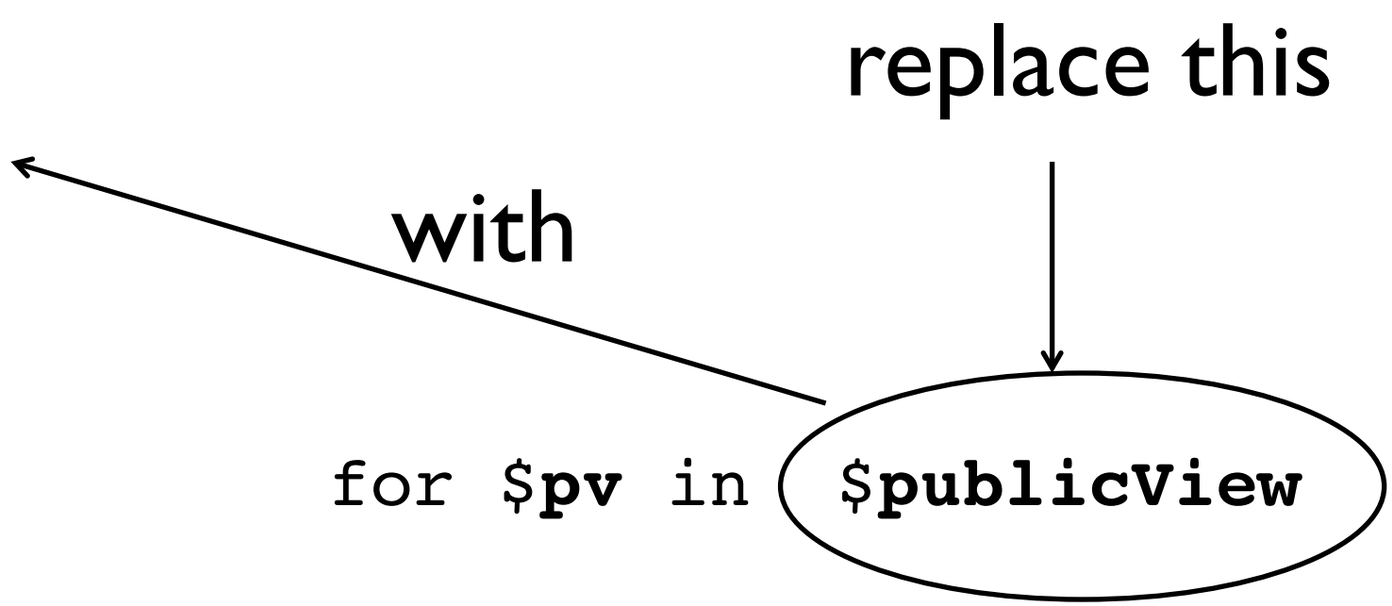
XQuery is Compositional !



To compute $(Q_{user} \circ Q_{admin})$

replace //`publicView` with Q_{admin} in Q_{user}

```
<publicView>{
let $cv:=//canonicalView
return

for $a in $cv/Actor
return

<actor>{(
$a/LName, $a/FName,

for $m in $cv/Movie

 where (
$c/Appears[ @aid=$a/@aid and
            @mid=$m/@mid ] )

return $m
)}<actor>
}</publicView>
```



Generally inefficient..

Query composition

Efficient query composition involves:

- substitution
- filtering
- pattern matching

To do this, the XQuery expression is first translated into XQuery-Core

Then, it is translated into a structural-tree and then simplified, in a functional style

Generating SQL

Theorem [Fernandez et al. 2002]
Any XQuery-Core expression over a canonical XML view of a relational database can be rewritten into an equivalent SQL/XML query in SilkRoute notation.

Silkroute

1. What the administrator sees : XML and DTD (the canonical view)
2. What the administrator writes : XQuery (the public-view query)
    1. this should be compliant with the SQL-annotated schema
3. What the user sees : XML (the public view)
4. What the user writes : XQuery (the user query)
5. What SilkRoute does with input XQuery $Q_{user} \circ Q_{admin}$: composition (tricky)
6. What SilkRoute does with the composed query : SQL conversion (tricky)

XML Export in Commercial RDBMS

DB2 User-defined mapping through DAD (Document Access Definition)

MS SQL Server 2005: Annotated schema (XSD): fixed tree templates; FOR-XML

Oracle 10g : SQL/XML, DBMS_XMLGEN (PL/SQL package)